ELC ONLINE
40 WEST 23RD STREET NY, NY 10010

Coding Requirements for Frontend Developers

VERSION
8.0

Prepared by	Katharine Jones, katjones@estee.com, 1 (917) 790-0181
Contributors	Jeremy Donelson, Greg Amoroso, Paul Gendek, Allison Messina, Laura Leslie, Jeff Kwok, John Hutchinson, Matt Schiffman
Version	8.0
Last updated	11/29/2018

Contents

Intro

This is a technical guide to onboarding new developers working on frontend code on ELC's Drupal Gem sites, including coding standard and requirements. These standards must be followed by all frontend developers writing code for Estee Lauder Online DrupalGem sites.

Onboarding

Developers must take all training courses and read all documentation as outlined in the "Developer" column of the training curriculum: https://elconline.app.box.com/v/Session-Training-Matrix

Technical Skills Checklist

	Responsibilities	Must Have Skills	Nice to Have Skills
CMS Content Editor	Non-technical user; updates content (images & text) within a CMS user interface	Experience working in a CMS	Basic HTML, understanding of SEO best practices
Maintenance Frontend Developer	Technical user; creates new templates in code or in the CMS interface; makes minor-intermediate front end updates	HTML5, CSS3, Javascript, jQuery,Sass Git	Drupal, Mustache, Lodash/Underscore
Advanced Frontend Developer	Very technical user; develops major new features and apps	HTML5, CSS3, Javascript, jQuery, Sass Git, basic understanding of Drupal hooks & theming	Drupal, Mustache Lodash/Underscore

Onboarding Requirements

Access Requirements:

- Drupal CMS user account with these roles:
  - Content Editor: creates and edits content
  - Content Approver: approves content so that it can be published
  - Content Publisher: publishes content to Production
  - Template Editor: creates and edits templates in the CMS
- Drupal CMS user account with access to GENIE and all other applicable Brands & Locales
- Alfresco: third-party platform that hosts all media assets on Drupal sites
- Jira: ticketing and project management tool
- Confluence: documentation library
- Personal server

* Access Requirements for Non-Technical CMS users
* Additional requirements for Front End developers

General Guidelines

YOU MUST ALWAYS:

- Follow the following community standards:
  - **HTML**
    - Mozilla Developer Network Web Developer Guides: https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5
  - **CSS/SASS**
    - Hugo Giraudel's Opinionated Style Guide for Writing Sane, Maintainable and Scalable Sass: https://sass-guidelin.es/
    - BEM Methodology: http://getbem.com/naming/
  - **JavaScript:**
    - Airbnb JavaScript Style Guide: https://github.com/airbnb/javascript/tree/es5-deprecated/es5
  - **Drupal**: https://www.drupal.org/docs/develop/standards
- Make all text translatable via the Drupal CMS user interface. This includes form error messages, page numbering, image alt tags, etc.
- Follow theme inheritance rules implemented in Phase 1 setup
- Correct any defects in code that you copy from elsewhere. (Do not assume that all existing code is correct.)
- Test on personal server before requesting Code Review
- Know when to use Drupal hooks js_alter and drupal_add_js. Consult the Drupal.org API documentation when using any hooks.
- Organize CSS styles in separate files, per brand convention
- Store data in a cookie or in local storage (vs making ajax calls)

DO NOT EVER:

- Hard code text, images, currency symbols, node IDs, or URLs
- Hard code JavaScript in Drupal files
- Leave extra spaces, tabs, etc.
- Attempt to change existing theme configurations
- Put images in the theme
- Add magicjs files via Drupal
- Make unnecessary Ajax calls to the PerlGem or Drupal servers
- Add redundant ajax calls on page-load to every page on a site where cookie/local or session storage can be used instead
- Hardcode any references to HTTP; all ELC Drupal sites are moving to secure (HTTPs) URLs.
- Use inline styles unless required to use a value from mustache (background-image, etc).

- Use theme or module hooks that directly reference node objects, global configuration variables, etc.
- Modify platform code, ie code in `/home/username/drupal/drupal-7.9/modules/` & `/home/username/drupal/drupal-7.9/sites/all/`
- Put CSS or JS code in the CMS or in Alfresco - it must be in code & follow our code review & release processes as outlined in this document

Codebase Overview

These are the key areas of the codebase that Frontend Developers will work:

`/home/username/drupal/drupal-7.9/sites/BRAND/themes/LOCALE/template_api/`
This is where custom locale-specific templates live.

`/home/username/drupal/drupal-7.9/sites/BRAND/themes/LOCALE/`
This is where custom locale-specific themes live. These themes files are for locale-specific overrides to the Brand's main theme & can be reviewed/released by regional teams.

`/home/username/drupal/drupal-7.9/sites/BRAND/domains/LOCALE-PREFIX.inc`
Locale-specific settings can be edited directly via the CMS "Site Wizard" before launch & in code post-launch.

`/home/username/drupal/drupal-7.9/sites/BRAND/themes/BRANDBASETHEME/`
This is where custom brand-specific theme lives & applies to all locales within the brand. All customization for the Brand need to reside in this sub-theme & must be reviewed/released by the Global Frontend Team.

Examples to Reference

GENIE, which stands for **G**eneric **EN**vironment for **I**nnovation and **E**volution, is a generic development site with a full suite of features that will be the source site for new brands, redesigns and new feature development. It also has fully functional account and check out, and its own product catalog.

GENIE Templates

The Global Front End Development team maintains a set of GENIE templates that can be used as examples for reference for developers building new templates:

- **Common templates:** `genie_base/template_api/common`

- **Editorial content templates:** `genie_base/template_api/editorial`
- **MPP and SPP templates**: `genie_base/template_api/products`

GENIE templates can be viewed within the codebase, through the CMS, or via a zip file if a developer needs to access them before gaining CMS access.

Smashbox Themes

Smashbox themes can be used as an example of best practice:

- `drupal-7.9/sites/smashbox/themes/smashbox_base`
- `drupal-7.9/sites/smashbox/themes/<locale>`

Accessing CMS Content on Sites You Don't Have Access To

To test example content updates or understand how content is constructed, you may access the CMS for a Brand or Locale that you don't have access to on your personal server only. This is safe & will never affect real live content. Follow these steps to grant yourself access on your personal server using "drush" commands:

To give yourself additional access rights on your personal server, please do the following:
`drush user-add-role "sys admin" --name="<username>"`

To give yourself access to additional locales on your personal server, please do the following:
`drush user-add-domain <local machine name> <username>`

Global Design Variants

Global Design Variants are business requirements specific to a locale or region, such as how the price per unit of measure for products are displayed. Variants are primarily implemented on templates and must be included for every locale and/or region so that content can be rolled out to other sites with minimal or no extra work.

Please refer to the current list of global variants in this document:

https://docs.google.com/spreadsheets/d/11fcECHrXha9xBIr0NUazEG4ilCE5P2SY9uIp2jDlfb8/edit#gid=903265682

Additionally, templates must include a text field for any text overlay on images. Images must never include hard coded overlay text, since content editors must be able to easily translate any text on localized content through the CMS.

Content Architecture

Drupal content that is used in more than one place on a site or app should be built once, and reused where needed via node references.

Media File Size Limits

Large images files are **bad for site performance**, and CMS users will be blocked from attaching images to content if the image file size is **500 kilobytes or higher**. Users are blocked from uploading new images to Alfresco via Drupal and also from attaching existing large image files that were uploaded directly through the Alfresco portal. A warning is shown to users before they attach an image **150 kilobytes or larger** on the PC tab of content, and **100 kilobytes or larger** on mobile tab.

For more information on ELC Online's Performance & Optimization Standards, see: https://elconline.box.com/s/5axy2zmahnup30dp9yh2m0pqitc2hgil

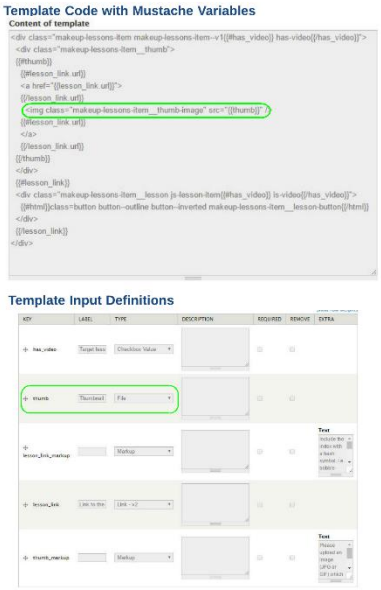
Image Optimizations

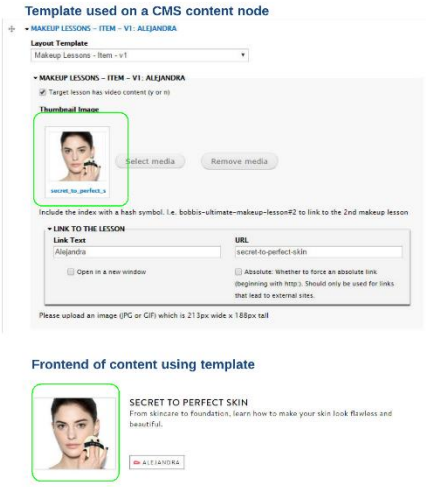
All images uploaded to Alfresco should be at a minimum losslessly compressed (ImageOptim, etc), and if possible, lossy compressed without introducing visual distortion/artifacts/blurriness/gain.

Drupal Templates

Templates allow developers to create custom layouts that can be added to various types of content. They can include a variety of input fields and are built with HTML, CSS and JS. Templates are entities defined by the Template Field contrib module.

The image below shows an example of how templates are constructed and used on content. Circled in green is an example of how an image is configured in a template, formatted by template code, updated by a CMS user, and rendered on the front end.





To see full image: https://elconline.box.com/v/template-code-example

REQUIREMENTS FOR ALL TEMPLATES:

- Work responsively on mobile or have a mobile alternative
- Follow the template naming conventions (see below)
- Have unique & accurate description
- Built so that images can be added through Alfresco
- Be assigned the correct tags

Template Tagging

Template tags determine what type of content it can be used on, whether it's a PC, mobile, or responsive template, if it's a partial, etc. Here are the general guidelines for template tags:

- Tags beginning with "device-" will enable the template to be used on pc, mobile, or both, content fields
- Tags beginning with "contenttype-" determine which content type(s) the template may be used on. The string value after "contenttype-" must match the machine name of the content type.
- "formatter" tags enable a template to be used as a formatter for all content types, and will not be used for the pc or mobile content fields
- Templates that are "partials" must use the "partial" tag

Go to **/admin/structure/template/add** to see the full list of possible template tags.

Template Naming Conventions

Label: {Descriptor} - {Device} - {version}

Machine Name: {descriptor}*{device}*{version}

Template labels & machine names **must not be longer that 255 characters**

Notes:

- {descriptor}: **required**. Value varies but describes template (i.e. grid, two column, etc). For Clinique this can be the "module name"
- {device} *optional*. Most templates should be responsive. Only needed if the template can only be used on Mobile or PC.
- {region or country}: *deprecated. Do not use this. Templates should be built with configurations so they can be used on any market.*
- {site type}: *deprecated. Do not use this. Templates should be built with configurations so they can be used on any type of site.*
- {version}: **required**. Used for each version of the template: v1, v2, v3, etc.

Examples:

Homepage Tout - v1 (homepage_tout_v1)

Store Locator - Mobile - v2 (store_locator_mobile_v2)

Two Column Scroller - v4 (two_column_scroller_v4)

Watch out for:

1. Template machine name should never begin with **"clone_of" - remove this** if you see it
2. **Do not put the Brand name in the template name**
3. Make sure the template name **ends with the version number**, with a lowercase "v", and there is no text after the number.

Partials

The mustache templating language allows developers to create template "includes" known as partials. Partial templates can be pulled into other templates so that the "partial" code isn't replicated unnecessarily.

If a template's .inc file includes an 'includes' array for partials, then the templates that it includes need to be tagged as a 'partial' in its 'tags' array.

Example

```
*parent_template_v1.inc*
'includes' => array(
 'partial_template' => 'partial_template_v1',
),
*partial_template_v1*
'tags' => array(
 'partial',
),
```

Mustache

Mustache is a simple template syntax. We use it in template code to represent content fields that are editable by CMS users.

Mustache best practices:

- Mustache conditionals - elements that don't have a value should usually not be shown on the page

- Use triple braces {{{ }}} to display unescaped html.

Wrong:

```
<div class="content">{{ content }}</div>
```

Correct:

```
<div class="content">{{{ content }}}</div>
```

- Never use triple braces inside of HTML attributes.

Wrong:

```
<span class="{{{ class }}}"></span>
```

Correct:

```
<span class="{{ class }}"></span>
```

- Use appropriate conditionals (where necessary)

Wrong:

```
<span>{{{ title }}}</span>
```

Or

```
<span>{{# title }}{{{ title }}}{{/ title }}</span>
```

Correct:

```
{{# title }}<span>{{{ title }}}</span>{{/ title }}
```

For more information on moustache, please read: http://mustache.github.io/mustache.5.html

Lazy Loading

- All <img> tags should utilize the lazysizes library, using data-src in favor of src in order to offload the request to the image
    - Lazysizes supports srcset for responsive images, and data-bg for CSS background images
    - Documentation: https://github.com/aFarkas/lazysizes

Please also read the Global Engineering team's Lazing Loading Guide For Frontend Developers

Template Caching

Except for rare exceptions, templates should not built to bypass caches since caching is needed for good site performance. If you have a question about whether your template should be cached or uncached, please escalate to the global engineering team.

Template Input Definitions

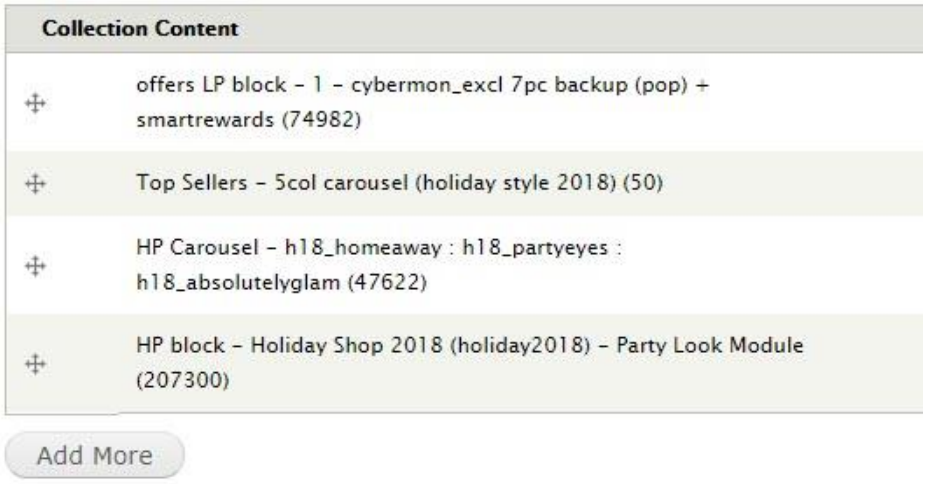
Use the right template input definition for the content that will be entered. Follow the guide at [Drupal Template Best Practices and Requirements]

Do not create new template input definitions, and do not use deprecated template input definitions.

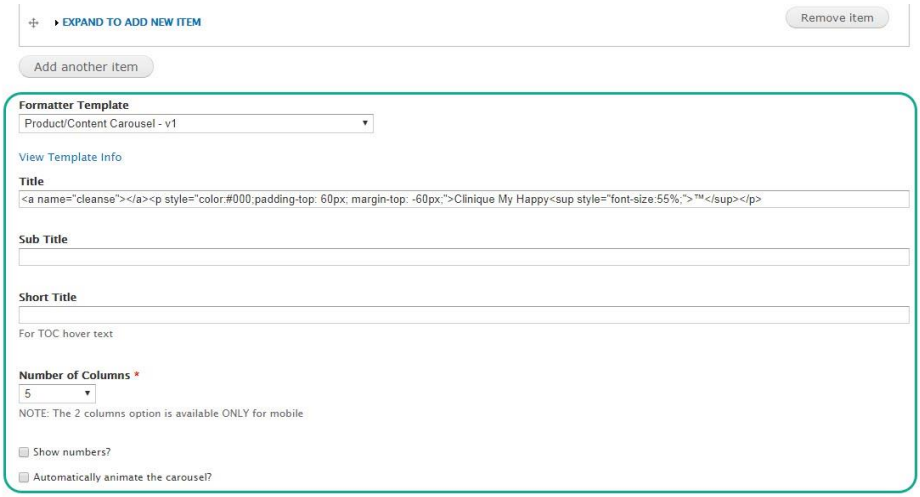
Formatter Templates

Formatter templates are used to control the layout around other templates. For example, a "carousel" formatter will output a set of templates in a rotating carousel format.

Formatter template on a Collection node:



Formatter template on a Block node:



Linting Tools

To ensure high quality of code, all frontend developers who work in our DrupalGem codebase are required to use our JavaScript / SCSS linting tools. We use ES Lint and SASS Lint with a set of rules that reinforces our coding standards.

This should be run in real time in your IDE while you are developing and can also be run with Gulp tasks on the server. It checks for common errors and helps ensure high quality code.

- **Developers must lint their code before they send it for code review**
- Code reviewers will reject new code that doesn't pass linting from being integrated, including into DEV & STAGE
- This is only for new code being developed - engineers are not required to refactor old code when they make changes to it (though they are encouraged to make improvements when possible!)

See linting rules here: https://elconline.box.com/v/lint-rules

Follow this guide for how to run the linting tools: Linting Guide for Frontend Developers

Javascript / Frontend

Javascript is used to handle user interaction and to communicate asynchronously with the PerlGem, Drupal, and 3rd party servers.

Checkout and account-specific Javascript lives in the PerlGem repository and follows that workflow. All other Javascript is in the Drupal repo.

Syntax

- Please indent with 2 space characters per tab. Do not use tab characters.
- Variables should be `camelCase`
- jQuery objects should be preceded with a "$" character, e.g. `$navLink`
- Use 1 space in front of parentheses, unless you are invoking a method.
- Do not use multi-line comments (`/* comment */`), even for a large block of code.
- Delete unused code. Don't just comment it out. If we need to retrieve it later, we can use source control.

Javascript Best Practices

- Do not pollute the global namespace with variables. Wrap your code in an IIFE and/or place your methods under namespaces.

Wrong

```
window.scrollHandler = function(e) {
        ...
};
```

Correct

```
(function(featureApp) {
        featureApp.scrollHandler = function(e) {
                ...
        };
})(window.featureApp || {});
```

- Do not check the length of an array on every iteration through the array. Store it in a variable instead.

Wrong

```
for ( var i = 0; i < myArray.length; i++ ) {
        // do stuff
}
```

Correct

```
var myLength = myArray.length;
for ( var i = 0; i < myLength; i++ ) {
    // do stuff
}
```

- Do not overuse jQuery's $() method. If you're going to reference a jQuery object more than once, take advantage of jQuery's support for chaining methods and/or store the object in a variable. Our convention is to name such variables with a prepended "$" character.

Wrong

```
$('.section-wrapper').addClass('lesson-section-'+sectionClass);
$('.section-wrapper').addClass('lesson-section-block');
$('.section-wrapper').css('overflow','auto');
$('.section-wrapper').css('display','none');
```

Correct

```
var $section = $('.section-wrapper');
$section.addClass('lesson-section-'+sectionClass);
$section.addClass('lesson-section-block');
$section.css('overflow','auto');
$section.css('display','none');
```

Correct

```
$('.section-wrapper')
   .addClass('lesson-section-'+sectionClass)
   .addClass('lesson-section-block')
   .css('overflow','auto')
   .css('display','none');
```

- Certain events, such as scroll and resize, get fired repeatedly in rapid bursts when a user initiates them. Handling that event every time it's fired is likely to impact browser performance. Prevent this by using debounce or throttle. Both are available in the underscore and lodash libraries, one of which should already be installed on all brand sites.

Wrong

```
$("body").on('scroll', function() {
 // Do expensive things
});
```

Correct

```
$("body").on('scroll', _.throttle(function() {
 // Do expensive things
}, 100));
```

Correct

```
$(window).on('resize', _.debounce(function() {
 // Do expensive things
}, 100));
```

- Select elements only by class names prepended with "js-" in your Javascript. These class names are reserved for Javascript usage and remain consistent between rollouts. Other class names are subject to change for styling purposes.

Wrong

```
<div class="account-header">My Account</div>
$('account-header').addClass('hidden');
```

Correct

```
<div class="account-header js-account-header">My Account</div>
$('.js-account-header').addClass('hidden');
```

SASS

Sass is a precompiler for CSS which adds nested rules, variables, mixins, selector inheritance, and more. Sass generates well formatted CSS and makes stylesheets easier to organize and maintain.

SASS is used across all of our sites. Each brand will have a “base” theme (example: estee_base) that contains styling information that spans all locales on that brand. Each locale “compiles” their own sass which inherits from the base theme, or a regional base theme (example: emea_base). A locale may add locale-specific overrides or variations to their Drupal subtheme’s sass files in order to customize the look & feel of their site.

To compile your locale’s sass files, please use the ‘`compile_css`’ command and enter the appropriate details.

Best Practices

- Do not hard-code hex values or font names. Check in the base theme to find variables instead. If you are adding a hex value that is not represented by a variable, add the variable.
- Do not nest your identifiers more than 3 levels deep (this can cause verbose CSS)
- SCSS code needs to be searchable. Do not overuse partial identifiers with the "&" character, as that makes it hard to search through source code when debugging. https://www.sitepoint.com/beware-selector-nesting-sass/
- Use $ldirection and $rdirection where supported instead of "left" and "right" when setting margin/padding/border/etc. This allows for easier right-justification for arabic language sites.
- @extend should only be used only pre-defined %rulesets. If you're unsure about whether or not to use @extend, check the compiled css output for huge definition blocks. If you're still unsure, just use a @mixin.

Caches

Drupal stores all its content in a database. Due to the complexity of our sites, each frontend page is assembled from dozens or hundreds of database queries. To make our sites fast for customers, Drupal sites have several layers of caching.

Engineers who can deploy code in Jenkins have access to clear caches as needed.

Drupal Cache

Engineers must clear these caches only after updating code and-or database on a personal server.

ELC Cache

Engineers must clear this cache only after updating a template in code or when troubleshooting template code.

Akamai

All public-facing Drupal pages are cached via Akamai to ensure they load quickly regardless of where in world the site visitor is located.

Important: Whenever releasing any code, release managers should verify immediately that the push was successful by bypassing Akamai and checking the site.

- Akamai will **clear automatically when content is published/unpublished** (including schedule-published).
- Allow up to **45 minutes** to see changes to the live site on **Personalization blocks** or content that is **global** (such as the footer nav)
- Allow up to **15 minutes** to see changes on the live site for **everything else**

Akamai can be cleared via Drupal UI at **/admin/config/system/Akamai**

Code Review & Escalation

Engineers - Follow these steps for successful code review!

- Follow the Drupal repo - code review checklists **before** doing a pull request and sending to peer review
- Make sure your code passes the linters: Linting Guide for Frontend Developers

- Test your own code - make sure it works!
- Make sure you have **two peer reviews that include comments in your PR** *before* you tag a senior reviewer
- Make sure you have the following items included
    - **Description in JIRA Ticket** - with explanation of the task, ideally user story and acceptance criteria.
    - **PR Description** - what are you doing & why
    - **Commit ticket numbers** using this exact format! "JIRA Issue Number: Commit Message" example "ABC-123: commit message"
    - **Candidate Branch name** must be stated in the JIRA ticket.
    - PR **links** back to the ticket.
    - JIRA ticket field - **Site Section** must be filled in if available
    - Verify that the **branch name** in bitbucket matches the candidate branch name in the jira ticket

- After peer review is complete - **Assign to the correct Senior Engineer** per the chart below. Please only tag ONE senior reviewer (unless you do you want reviews from more than one - if so please tag them & comment this). Don't just tag the first reviewers in the list – **choose at random** so the reviews are more evenly distributed.
- Only the senior code reviewer or the implementation engineer is **allowed to merge a PR**. You must **NOT** merge a PR until you have approval (green checkmark) from a senior code code reviewer! If you have tagged extra (unnecessary) senior code reviewers, you must remove them from the PR before you merge it.
- If you want to expedite **.inc files**, put them in their own branch & their own ticket. The ticket can be a sub-task. (They must follow the same workflow as other frontend code, but it will go faster in its own branch & ticket.)
- Senior reviewers will review within 24 hours. **Please do not ping them to ask them to review a PR.** If more than 24 hrs have passed, contact the "Point of Contact" listed in the table below.

***Note**: If you assign to the wrong engineer or are missing any item outlined above, IT WILL CAUSE A DELAY!*

If you do not follow all of these steps, the senior code reviewer can reject your PR.

REMINDER - ONLY assign to a senior reviewer AFTER passing at least two peer reviews. Do not add the Senior until the Peer review is complete.

Once your code is approved it will follow the frontend release process.

Senior Engineering Review Assignments

Area	Description	Examples	Senior Code Reviewer	Backup (special request only)	Point of Contact for Escalations & Prioritization
Content	Editorial & Gnav templates	Content carousels, Boutiques, header & footer navigation - most frontend code that doesn't have a backend integration fits here	Neal Heneghan Suzi Arnold Cory Lashway Michal Slepko	Deanna Earley Alex Staples	Allison Messina
Checkout & Account	Any frontend code in the Drupal repo that affects Checkout & Account	Facebook login, content & product templates that are used on checkout & account pages	John Hutchinson Jonathan Hall Michal Slepko	Deanna Earley Alex Staples	Ari Parikh
Product Features	Templates that display products & code related to how product data is consumed & displayed by the frontend	Templates for MPP, SPP, product touts; template input definitions that pull in products	Greg Amaroso Neal Heneghan Cory Lashway	Deanna Earley Alex Staples	Allison Messina
Analytics	Any analytics integration	Tealium, etc	Lucas Healy Pat McErlean		Alex Joachim
Store Locator	Store locator frontend & code related to how store data is consumed & displayed by the frontend	Store locator templates	Greg Amaroso *backup: Mattie Currie*		Allison Messina
Promotions	Code related to promotions	GWP templates, templates that integrate with offers & offer data	Chris Petrunis	Alex Staples	Allison Messina
Search	Code that displays the frontend of search & that integrates with search tools	Search results, Search box	Pat McErlean Greg Amaroso		Allison Messina

Drupal CMS Backend Platform	The core Drupal code that handles CMS functionality & frontend page rendering	anything in sites/all	Christopher Donnelly Jeff Kwok		Montse Cebrian
Omni	Omnichannel projects	BORIS, BOPUS, Loyalty	Chris Dial Mattie Currie Michal Slepko		Allison Messina
Frontend Foundations	Core frontend technology that is used across the platform	New technology integrations & pilots, performance & modernization initiatives, gulp files, security, Drupal code for React integration into GENIE (including mustache files for React components)	Paul Gendek		Matthew Homeijer

Region & Coresite Specific			**APAC**: *(TO BE ADDED)* **EMEA**: Deanna Earley Chris Petrunis Michal Slepko Alex Staples **LATAM** : Eric Dillon **NA**: *(FOR NOW USE GLOBAL)* **UK**: Paul Holmes Dom Hastings Matthew Jackson Alexander Yezhov Ajith Kumar **Coresite base themes:** *(TO BE ADDED)*		

Senior Reviewers

	Senior Code Reviewer	Time Zone
1	Greg Amaroso	EST
2	Suzi Arnold	
3	Mattie Currie	
4	Chris Dial	
5	Christopher Donnelly	
6	Deanna Earley	BST
7	Paul Gendek	EST
8	Jonathan Hall	
9	Lucas Healy	
10	Neal Heneghan	
11	John Hutchinson	
12	Jeff Kwok	
13	Cory Lashway	
14	Pat McErlean	
15	Chris Petrunis	
16	Michal Slepko	BST
17	Alex Staples	BST

Escalate to Drupal Platform Team via drupal@estee.com

- Any code in the Drupal platform-level codebase
- Requests for new content types, even if they apply to only one brand (this is not recommended and should be done via templates instead)
- Code that includes queries that affect the Drupal database
- Any major architectural changes that affect the ways sites are rendered or how code is released
- Any new modules, overrides to existing modules, user interface changes, configuration pages, or permissions
- Any changes to how node references work, such as extended or new node reference input types (these will break localization if done incorrectly)

Additional Support

- For Drupal platform-related questions/requests, email drupal@estee.com
- To report CMS slowness: http://goo.gl/forms/1lWxwTcvyr
- To report personal server, DEV & STAGE slowness: IOT - Systems (IOTR)
- If your updates aren't appearing on the live site request support for caches: http://goo.gl/forms/a7qgdRcIIc
- Login issues: https://jira.esteeonline.com/secure/CreateIssue.jspa?pid=10700
- Slack is the quickest way to get answers:

Topic	Room Name	Link
General Technical Q&A	#eng-gen-discussion	https://elconline.slack.com/messages/CE2NLE71Q/details/
Bitbucket branching strategy, GIT, code review & deployment workflow	#eng-fe-workflow	https://elconline.slack.com/messages/CE24AKETB/details/
Questions for engineering leadership team	#eng-mgmt	https://elconline.slack.com/messages/CE3QAKQ0P/details/

Escalations & questions related to the core Drupal application & CMS	#eng-drupal-platform	https://elconline.slack.com/messages/CE20YSLU8/details/
Escalations & questions related to the Frontend	#eng-frontend	https://elconline.slack.com/messages/CE2UR8JER/details/
Onboarding for new engineers	#eng-onboarding	https://elconline.slack.com/messages/CE3UDJZM0/details/
Local development questions & discussion (currently in pilot)	#eng-local-dev	https://elconline.slack.com/messages/CE44TAY2H/details/
Escalate & discuss live site issues	#live-site-issues	https://elconline.slack.com/messages/CEA7MDE90/details/

Appendix A: Code Review Checklist

For frontend code in the Drupal codebase

General

- Make sure ALL GIT commits include the associated JIRA ticket number
- Make sure the Jira ticket has the name of the branch that's getting reviewed
- Make sure the PR is linked back to the Jira ticket
- Confirm it passes linting / Reject it if it doesn't - *[link]*
    - JavaScript
    - SCSS

- Branch should pass global `gulp qa-js-diff` command (*checks diffs in branch for JS linting errors)*
- Ensure none of the updated files contain DOS line endings

Javascript

- DO NOT use the eval() function EVER!
- .scroll() and .resize() functions should always be debounced/throttled
- Do not repeatedly query for the same element; do it once and store it in a variable.
- Do not embed event handlers within each other. For example: do not put a window.load event inside a document.ready
- JS event handlers should bind to elements with "js-" prefixed classes
- **Anticipate hoisting** *[link]*
- avoid recasting of datatypes - e.g. *var obj = ''; and then later... obj = {foo:bar};* is not acceptable.
- variable names should be consistent - use $ in front of all variables names that are jQuery selectors
- reject highly complex conditionals. suggest either breaking out into functions or use an an object lookup if appropriate
- nested loops are bad, there is usually a better way
- Avoid global variables as much as possible, try to always ensure code is encapsulated (IIFE, Class) with global references passed as parameters.

SCSS/CSS

- There should be no selectors with the "js-" prefix in any scss file
- All media query breakpoints should use global variables
- All fonts and colors should use global variables
- Ensure that RTL support is included (not applicable for all brands)
    - when setting shorthand horizontal margin or padding please use the swap_direction mixin (if the left/right values differ)
- Write styles "mobile first"
- **avoid usage of @extend**
    - do NOT introduce new (we hope to deprecate due to widespread improper usage).
    - If @extend is used, NEVER extend classes directly as `@extend .foo` – only silent classes are ever permitted as `@extend %foo`

ADA

- All images should have an "alt" tag specified, even if it is blank

New Templates

This should be a first pass for peer review.

- Templates should be properly named *[link]*
  - *Label: {Descriptor} - {Device if PC/mobile specific} - {version}*
  - *Machine Name: {descriptor}{device if pc/mobile specific}{version}*
- CSS/SASS naming should adhere to BEM standards *[link]*
- Class names only have 1 element, therefore only 1 "__" per class
- Folder structure should follow conventions per brand (templates & css)
- Mustache syntax must be properly used *[link]*
  - *Double braces : Escaped variables*
  - *Triple braces : Unescaped HTML content*
  - *Elements that don't have a value should usually not be shown on the page*
  - More examples at Coding Requirements for Front End Developers: Live Document#Mustache

- All images must be lazy loaded
- JS classes should be included in the template folder and named for the template (ie. fancyTemplate.js)
- All libraries should be properly defined / attached
- Ensure proper usage of Drupal behaviors / jQuery events (no outside methods)
  - attach method should pass 'context' parameter
  - all jQuery selectors should filter by template context
- No t() functions *(deprecated)*
- No free text class / id fields; all of the possible options should be added to a select list
- Use url_v2 or link_v2 fields for URLs or links, respectively. Do not use string fields.
- Templates should not use the input types flagged as "**Not Recommended**" in the list below:

Not Recommended - DO NOT USE	Use Instead
category	product
checkboxes	checkbox_value
dimension	select_v2
elc_empty_template	n/a
fieldset	component_group
hidden	n/a
number	select_v2
sku	product
text	elc_text_format
text_format	elc_text_format
vertical_tabs	n/a
video	internet_video

- Do **NOT** add new brand-level template input definitions. All future template input definitions must be global so that templates can be used on multiple brands.
- Make sure they are not creating a new template when they could just add a configuration on an existing template.
- Product input query key guidelines: for performance, if a new product tout is created, create a new querykey for that tout rather than using MPP/SPP one that is bigger than needed

Drupal module files or php (including .inc files)

- Run PHP files through a CLI parser. Any errors should result in code review being failed. Example: `php mymodule.php`

Internationalization

- No hardcoded text (All content should be included in the translation set / template form) <=== **if you do this, regional engineers will hunt you down and make you suffer.**

Third Party Scripts

- Check the size of third party scripts & reject any that will impact performance beyond acceptable standards